GUI for LCDs

V. 1.0

www.alexbelectronics.com

CONTENTS

Introduction

This library gives wide range of opportunities for embedded application developers to make a user-friendly interface of any kind for any type of applications, using black/white small LCD panels with or without touchscreen and a control module with minimal system abilities (look at hardware requirements). You can create completely any graphics without loosing much time for drawing texts, diagrams, buttons and other controls, pictures etc concentrating your affords on the main functional goal. The library doesn’t use LCDs in text mode, and that gives more flexibility. What you have to do for adopting the library to your LCD is to rewrite the CommandWrite and DataWrite and DataRead functions which are specific for your LCD module.

1.Hardware requirements and limitations

The recommended controller features are:

8 bit processors are good
24 Million Instructions Per Second
64-128K Program flash memory
4 K RAM (64 K Preferred)

The GUI_main.c module contains all graphic functions and requires:

Near 15K of code space
Near 700 bytes of RAM space
DrawGraph() function requires floating-point calculations, sprintf() standard function (stdio library)
Some functions require strlen() standard function (string library)
DataRead(), DataWrite(),CommandWrite() function use device specific registers to write data to I/O port. The present example uses c8051f120.h SFR registers.

The events.c module contains events mechanism, to make it easy to interact with user and requires:

Near 300 bytes of code space
Near 900 bytes of RAM space

The touchscreen module given as an example of proceeding with MK 715 touchscreen controller, requires:

Near 300 bytes of code space

The LCD module: Must have graphical mode. Each bit corresponds to the pixel on the screen. The memory must be byte addressable and [0000] byte data must correspond to 8 top-left pixels of LCD, [0001] byte must correspond to the next 8 pixels to the right in the same horizontal line.

2. Differences from ANSI C

- Using memory type specifiers (xdata – external RAM space, bit – internal bit-addressable RAM, code – constants in code space)
- You can read more from c51.pdf, lib51.hlp from KEIL ELEKTRONIK GmbH

3. Functions description

All functions in the present library are divided onto:

- Graphics functions
- Control functions & Text functions
- System functions

3.1 Graphics functions

For all graphical functions, the point of origin is top-left corner of the screen if LCD video memory lowest address corresponds to the top-left pixels.

void DrawBMP(unsigned char coord_Y, unsigned int coord_X, unsigned int width, unsigned char height, unsigned char* icon_ptr, unsigned char status, unsigned char event_flags, unsigned char object_id);

Action: Draws binary black/white bitmap in <coord_Y>, <coord_X> top left corner coordinates with given <width>, <height>. Icon_ptr – points at the beginning of the bitmap array.

Status byte defines the action on the bitmap. Following flags can be applied
SECOND_SCREEN_PLANE – draws to second screen plane. Second screen plane memory offset must be specified in Gui_lcd.h. By default function draws to the first screen
FILLEDAREA – fills the rectangular, area specified by the coordinates and width, height with black or transparent pixels depending on HIGHLIGHT flag. The icon pointer is ignored.
HIGHLIGHT- can be used with FILLEDAREA (FILLEDAREA|HIGHLIGHT) to set the pixel to on state within the area. Otherwise the area will be cleared.
Event_flags defines the type of visual effect, that appears when the bitmap is touched (if events are enabled and touchscreen panel is used) this flag must be processed in touchscreen interrupt service routine.(see interrupts.c)
Object_id specifies the id number for the bitmap object **(**if events are enabled and touchscreen panel is used). The maximum 100 objects can be available for touch on the screen. The maximum value of object_id is 254, but the quantity of objects simultaneously must not exceed 100. Specify 0 if the object must not be active for touch. Quantity of objects with object_id=0 is not limited.

void DrawPixel(unsigned int coord_X, unsigned char coord_Y, unsigned char status);

Action: Draws pixel in the specified coordinates
Status:

SECOND_SCREEN_PLANE – draws to second screen plane. Second screen plane memory offset must be specified in Gui_lcd.h
SWITCHOFF – if specified, function turns off the pixel, otherwise – highlights the pixel

void DrawLine (unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, unsigned char status);

Action: Draws line between two points starting from <coord_X1, coord_Y1> to <coord_X2, coord_Y2>.
Status:

SECOND_SCREEN_PLANE – draws to second screen plane.
SWITCHOFF – if specified, function turns off the pixels, by default or if HIGHLIGHT is specified – highlights the pixels;
DOUBLEWIDTH - Draws the line with double width (by default – single pixel width)
TRIPLEWIDTH – Draws the line with triple width

void DrawRectangle(unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, unsigned char status);

Action: Draws rectangle with top-left corner in <coord_X1, coord_Y1> and bottom-right corner in <coord_X2, coord_Y2>
Status:

SECOND_SCREEN_PLANE – draws to second screen plane.
SWITCHOFF – if specified, function turns off the pixels, by default – highlights the pixels
DOUBLEWIDTH - Draws the rectangle with double width (by default – single pixel width)
TRIPLEWIDTH – Draws the rectangle with triple width

void DrawFilledRectangle(unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, unsigned char status)

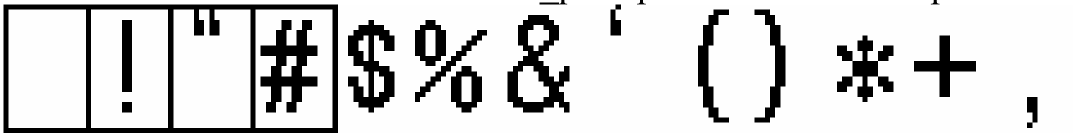
Action: Draws filled rectangle with top-left corner in <coord_X1, coord_Y1> and bottom-right corner in <coord_X2, coord_Y2>

Status:
SECOND_SCREEN_PLANE – draws to second screen plane.
HIGHLIGHT – Fills the Rectangle with highlighted pixels, by default erases the rectangle area;

3.1 Control & Text functions.

void SetCurrentFont(unsigned char code *font_ptr, unsigned char fnt_size_w, unsigned char fnt_size_h, unsigned char chr_count, unsigned char first_chr_offset, unsigned char chr_density);

Action: initializes current font. font_ptr – pointer to the bitmap in code memory:



the bitmap must be stored in memory rasterizing from left to right line by line;
<fnt_size_w> – width of the character cell in pixels,< fnt_size_h> – height of the character cell in pixels;
<chr_count> – total number of characters;
<first_chr_offset> – if You provide not a full character set (excluding for example special symbols from 0x00 to 0x20) you can specify this constant to calculate the real character code.
<chr_density> - space between characters for text output in pixels;
Example: SetCurrentFont(&font24x14[0], 16, 24, 224, 0x20, 2);
Sets the font pointed by font24x14 with 16 pixels character width and 24 pixels height, total number of symbols is 224, 0x20 first symbols skipped, the space between characters when outputting the text will be 2 pixels.

unsigned int DrawText (unsigned int X, unsigned char Y, unsigned char* str_ptr, unsigned char length, unsigned char status, unsigned char event_flags, unsigned char object_id);

Action: Draws the string pointed by <str_ptr> with <length> number of characters, in coordinates <X>, <Y> (top-left corner or top-right corner depending on status byte). Function returns the total length of the printed string in pixels. **SetCurrentFont() must be applied before.**
Status: SECOND_SCREEN_PLANE – draws to second screen plane;
RIGHT_ALIGN – <X>, <Y> specify top-right corner of the text, by default – top-left corner;
SMART_FONT – provides a good possibility to print the text in “windows” like mode. If that flag enabled the function scans through each character and finds left and right offsets to character highlighted pixels within the character cell thus the function prints not the whole char cell but only occupied area. The text printed with SMART_FONT looks tiny.
Event_flags defines the type of visual effect, that appears when the text area is touched (if events are enabled and touchscreen panel is used) this flag must be processed in touchscreen interrupt service routine.(see interrupts.c)

Object_id specifies the id number for the bitmap object **(**if events are enabled and touchscreen panel is used). The maximum 100 objects can be available for touch on the screen. The maximum value of object_id is 254, but the quantity of objects simultaneously must not exceed 100. Specify 0 if the object must not be active for touch. Quantity of objects with object_id=0 is not limited.

void DrawIconButton (unsigned int coord_X, unsigned char coord_Y, unsigned char* caption_ptr,unsigned char cap_length, unsigned char size, unsigned char* icon_ptr, unsigned char status, unsigned char event_flags, unsigned char object_id)

Action: Draws a button with icon picture, with top-left corner in **coord_X, coord_Y**
    <caption_ptr> - button caption string;
    <cap_length> -caption length, if cap_length=0, no text will be printed;
    <size> - defines the space in pixels between consistent elements of the button, thus the size can be ajusted;
    <icon_ptr> - pointer to the icon bitmap; icons must be square; the icon size is defined in Gui_main.c as SYSTEM_ICON_SIZE;
Status : SMART_FONT – see DrawText(); other flags are not aplicable; function draws in the main screen plane only;
Event_flags: defines the type of visual effect, that appears when the button is touched (if events are enabled and touchscreen panel is used) this flag must be processed in touchscreen interrupt service routine.(see interrupts.c)
Object_id: specifies the id number for the object **(**if events are enabled and touchscreen panel is used). The maximum 100 objects can be available for touch on the screen. The maximum value of object_id is 254, but the quantity of objects simultaneously must not exceed 100. Specify 0 if the object must not be active for touch. Quantity of objects with object_id=0 is not limited.

void DrawButton (unsigned int coord_X, unsigned char coord_Y, unsigned char* caption_ptr, unsigned char cap_length, unsigned char size, unsigned char status, unsigned char event_flags, unsigned char object_id);

Action: Draws a button with top-left corner in **coord_X, coord_Y**
    <caption_ptr> - button caption string;
    <cap_length> -caption length, if cap_length=0, no text will be printed;
    <size> - defines the space in pixels between caption and button borders, thus the size can be ajusted;

Status : SMART_FONT – see DrawText(); other flags are not aplicable; function draws in the main screen plane only;
Event_flags: defines the type of visual effect, that appears when the button is touched (if events are enabled and touchscreen panel is used) this flag must be processed in touchscreen interrupt service routine(see interrupts.c)

Object_id: specifies the id number for the object **(**if events are enabled and touchscreen panel is used). The maximum 100 objects can be available for touch on the screen. The maximum value of object_id is 254, but the quantity of objects simultaneously must not exceed 100. Specify 0 if the object must not be active for touch. Quantity of objects with object_id=0 is not limited.

void DrawGraph(unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, unsigned int start_X, unsigned int end_X, unsigned char xdata* Graph_record_ptr, unsigned char gr_type, unsigned char event_flags, unsigned char object_id)

Action: Draws graph window with top-left corner in <coord_X1>, <coord_Y1> and bottom-right corner in <coord_X2>,<coord_Y2>. To draw the graph it is necessary to turn it into the following structure:

```
 xdata struct graph_record
{

        float wavelength;
        float absorbance;

};
 xdata struct  graph_data
{
        unsigned int Min_X_view;
        unsigned int Max_X_view;
        unsigned int Marker_X_index;
        struct graph_record data_array[]; //specify the maximum number of points
};
```

where <Min_X_view> - left point index; Max_X_view – right point index;
<Marker_X_index> - marker index if necessary;
<data_array> - data array of points

The function scans the data array from <Min_X_view> index to < Max_X_view > index points, finds the biggest value for Y coordinate to adjust the vertical axis. It draws the points depending on gr_type byte as broken line or as dots. You can draw several graphics buy locating them one after another in the data_array and setting Min_X_view on the first point of first graph data and Max_X_view on the last point of last graph. The function searches for the data_array[i].X and when another graph starts (when next X value lower than previous one) changes graph type from DOT_DIAGRAM to LINE DIAGRAM.

<Gr_type>: NUMBERS_OFF – cancels limit values output for both axis;
FRAME_OFF – cancels border lines output;
DOT_DIAGRAM – dotted diagram output;
LINE_DIAGRAM – broken line type of graph;

Event_flags: defines the type of visual effect, that appears when the button is touched (if events are enabled and touchscreen panel is used) this flag must be processed in touchscreen interrupt service routine.(see interrupts.c)
Object_id: specifies the id number for the graph object **(**if events are enabled and touchscreen panel is used). The maximum 100 objects can be available for touch on the screen. The maximum value of object_id is 254, but the quantity of objects simultaneously must not exceed 100. Specify 0 if the object must not be active for touch. Quantity of objects with object_id=0 is not limited.

unsigned char DrawMarker(unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, float X, unsigned int scan_X_min, unsigned int scan_X_max, unsigned char * Graph_record_ptr, unsigned char status)

Action: Draws marker for a graph point using graph area information: top-left corner <coord_X1>. <coord_Y1>; bottom-right corner <coord_X2>, <coord_Y2>;
Searches for <X> value in <Graph_record_ptr> graph record in the limits of index from <scan_X_min> to <scan_X_max>
Status: HIGHLIGHT – if specified the marker will be drawn;
SWITCHOFF- the marker for the specified point will be erased;
SECOND_SCREEN_PLANE – draws marker to second screen plane;

3.3 System functions

void ClearScr (unsigned char status);

Action: clears the screen area in specified screen plane calculating screen parameters from defined LCD_width and LCD_height in GUI_main.c;
Status: SECOND_SCREEN_PLANE – clears second screen plane

Note: Clears the LCD memory directly using CommandWrite() and DataWrite() that must be corrected for the used particularly LCD module in case VIDEO_RAM_ENABLE is not defined. If VIDEO_RAM_ENABLE defined clears VideoRAM[] array in the controller external RAM.

void CommandWrite (unsigned char byteInd);

Action: Writes <byteInd> byte to controller port which LCD module is connected to, Sets correspondent port bits to specify command write operation. In GUI_main.c it is as an example for SED1335 LCD controller and analogs (please see sed1335.pdf description);

void DataWrite (unsigned char byteInd);

Action: Writes data byte <byteInd> to the apropriate controller port for LCD_module; sets the control bits to specify data write operation. In GUI_main.c it is as an example for SED1335 LCD controller and analogs (please see sed1335.pdf description);

unsigned char DataRead (void);

Action: Returns data byte from LCD_module. Appropriate command must be applied before reading bytes from LCD_module. In GUI_main.c it is as an example for SED1335 LCD controller and analogs (please see sed1335.pdf description);

void Delay(unsigned int num);

Action: Makes a delay**.** Must be adjusted for the specific controller clock rate;

void InitLCD (void);
Action: Sends set of commands to LCD module for initialization of graphical mode and three screen planes, sets the overlapping logic for the planes In GUI_main.c it is as an example for SED1335 LCD controller and analogs (please see sed1335.pdf description);

void ResetLCD(void);

Action: sets reset bit to high for LCD module initialization;

void RefreshScreen(unsigned char status);

Action: Draws VideoRAM[] data to LCD module via CommandWrite(), DataWrite(). The function is defined only when VIDEO_RAM_ENABLED;
Status: SECOND_SCREEN_AREA – redraws second screen area;

void RefreshScreenArea(unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, unsigned char status);

Action: Draws VideoRam[] area correspondent to rectangular window with top-left corner in <coord_X1>, <coord_Y1>, and bottom-right corner in <coord_X2>, <coord_Y2> to LCD_module via CommandWrite(), DataWrite(). Function redraws the rectangular area with coordinates divisible by 8;
Status: SECOND_SCREEN_PLANE – refreshes second screen area;

void CopyVRAMarea(unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, unsigned char status, unsigned char * dest_ptr);

Action: Copies VideoRAM[] area specified by screen coordinates to destination array specified by <dest_ptr>
Status: SECOND_SCREEN_PLANE –copies from second screen plane of VideoRAM[] area;

void RestoreVRAMarea(unsigned int coord_X1, unsigned char coord_Y1, unsigned int coord_X2, unsigned char coord_Y2, unsigned char status, unsigned char * source_ptr);

Action: Loads VideoRAM[] area with specified screen coordinates from array specified by <source_ptr>

Status: SECOND_SCREEN_PLANE –Loads to second screen plane of VideoRAM[] area;

void ClearScrVRAM(unsigned char status);

Action: Clears VideoRAM[] area of the whole screen plane, specified by status;

Status: SECOND_SCREEN_PLANE –clears second screen plane;

4. Compiling options.

If (#define VIDEO_RAM_ENABLE) – module is compiled to draw graphics to RAM memory. If there is no such definition – al graphics will be drawn directly to LCD module.

First screen area by default is defined to start from 0x0000 address. Second screen area address depends on screen size and must be defined:
#define SECOND_SCREEN_PLANE_OFFSET <second_plane_start_address>

All commands for LCD module are defined for SED1335 controller type LCDs and must be redefined for other controllers. The example given below (most of the functions are used to make presets for this particular LCD. For drawings MREAD MWRITE, CSRW memory access commands only used):
```
//SED1335
        #define SYSTEM_SET  0x40
        #define SLEEP_IN    0x53
        #define DISP_ON     0x59
        #define DISP_OFF    0x58
        #define SCROLL      0x44
        #define CSRFORM     0x5D
```

```
#define CGRAM_ADR   0x5C
#define CSRDIR_RIGHT  0x4C
#define CSRDIR_LEFT 0x4D
#define CSRDIR_UP   0x4E
#define CSRDIR_DOWN 0x4F
#define HDOT_SCR    0x5A
#define OVLAY       0x5B
#define CSRW        0x46
#define CSRR        0x47
#define MWRITE      0x42
#define MREAD       0x43
```

if (#define EVENTS_ENABLE) –predefinition is used events are enabled. All functions that have object_id and event_flags arguments will create event record using events.c module (see events description in Events engine section).

#define SYSTEM_ICON_SIZE <size> - must be defined if system icons are used for buttons. 32 pixels icons are used in the source code

#define LCD_width <width_in_pixels>
#define LCD_height <height_in_pixels>
this definitions are necessary for all graphical calculations

5. Events engine.

Events engine provides a set of functions to organize users interface using touchscreen panel. If an element (for example a button or Text) was drawn using object_id>0 an event record will be created. After building user's screen the programm rotates in neverending loop and calls GetLastEventObjectId(). If 0xFF returned – events buffer is empty and no events occurred, otherwise if the screen was touched function returns object_id of the object which screen area was touched. It is necessary to write your own code for interrupts and touchscreen processing. Given example uses MK715 controller and calculates screen coordinates for 320x240 LCD.
InitEvents(void) –initalizes events buffers;
SetEvent(unsigned char xdata *e_rec) – sets event for the object using event_record :

```
xdata struct event_record
{
        unsigned char object_id;
        unsigned int X1;
        unsigned char Y1;
        unsigned int X2;
        unsigned char Y2
        unsigned char flags;

};
```

X1, Y1 – top left corner of the sencitive zone;
X2,Y2 – bottom right corner of the sencitive zone;

Flags – type of action for displaying the reaction on touch. (in the example three type of actions are defined:

```
        #define HIGHLIGHT_SENSITIVE_ZONE 0x80
        #define SWITCHOFF_SENSITIVE_ZONE 0x20
        #define LEAVE_HIGHLIGHTED 0x40
```

unsigned char GetLastEventObjectId(void) – returns the object id for the last appeared event;

unsigned char xdata* GetLastEventRecord(void) – returns the pointer on the last appeared event record;

6. Touchscreen support.

Use touchscreen.c module to make touchscreen support. The example is given for MK715 controller.
unsigned int Read_n_Write_touchscreen(unsigned char addr, unsigned char byte) – reads/writes data of MK715
Use interrupts.c to sequentially process data from touchscreen controller with interrupt routine. The interrupt routine compares the returned coordinates of touch spot and compares to the zones of the events records. If the spot appears within the touch-sensitive zone it places the event record to events_search_pull. When the main function requests GetLastEventObjectId(), it gets the last event object. The event observed to be processed. If 0xFF returned – events buffer is empty and no events occurred.